Implementing $\mathcal{X}$

Steffen van Bakel and Jayshan Raghunandan

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK

Abstract

This paper presents a term graph rewriting system as an implementation for the $\mathcal{X}$-calculus, a substitution-free language that can be used to describe the behaviour of functional programming languages at a very low level of granularity, and has first been defined in [10,1]. Since the calculus has a notion of binding, in the context of sharing a notion of rebinding is introduced that helps avoid double binding of names.

Introduction

This paper will present a notion of term graph rewriting as model of implementation for the (untyped) calculus $\mathcal{X}$, as first defined in [10] and later extensively studied in [1]. The origin of $\mathcal{X}$ lies within the quest for a language designed to give a Curry-Howard correspondence to the sequent calculus for Classical Logic, in particular in [12]. $\mathcal{X}$ is defined as a substitution-free programming language that, perhaps surprisingly, is extremely well-equipped to describe the behaviour of functional programming languages at a very low level of granularity, and has been defined in [10]. Its relation with the Lambda Calculus and $\lambda\mathbf{x}$, the calculus of explicit substitutions, was studied in detail in [1].

The Curry-Howard property strongly links typeable programs and provable theorems, and can be understood as follows:

> (CURRY-HOWARD ISOMORPHISM) "*Terms as Proofs, Types as Propositions.*" Let M be a term, and A a type, then M is of type A if and only if A, read as a logical formula, is provable in the corresponding logic, using a proof which structure corresponds to M.

[1] Email: svb@doc.ic.ac.uk, jr200@doc.ic.ac.uk

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

A sequent style implicative classical logic can be defined by:

$$(Ax) : \frac{}{\Gamma, A \vdash \alpha{:}A, \Delta} \qquad (cut) : \frac{\Gamma \vdash \alpha{:}A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$$

$$(LI) : \frac{\Gamma \vdash \alpha{:}A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A{\to}B \vdash \Delta} \qquad (RI) : \frac{\Gamma, A \vdash \alpha{:}B, \Delta}{\Gamma \vdash A{\to}B, \Delta}$$

In this isomorphism, there exists a strong relation between reduction in the calculus of terms, and normalisation of proofs, also known as *cut*-elimination.

As far as the Curry-Howard isomorphism is concerned, $\mathcal{X}$ stands out in that it is the first calculus to achieve that in full for a classical logic. For example, in $\overline{\lambda}\mu\tilde{\mu}$, all provable propositions can be inhabited, but not all terms correspond to proofs, and in $\lambda\mu$, not all proofs can be represented, since there reduction is confluent. In contrast, cut-elimination in implicative classical logic is not.

Starting from different approaches in that area [9,12], in [10] the calculus $\mathcal{X}$ was introduced, and shown to be equivalent to the $\overline{\lambda}\mu\tilde{\mu}$-calculus of [9] in terms of expressivity. Using this correspondence, a strong normalization result was shown for $\overline{\lambda}\mu\tilde{\mu}$. In fact [10], as well as [12], did not study any property of *untyped* $\mathcal{X}$, but focused only on its type aspects in connection with the sequent calculus; [12] set out to study the structure of proofs, so the terms there carry types and correspond to proofs. In $\mathcal{X}$, we study terms without types, and drop the condition that terms should represent proofs of the logic altogether: we study a pure calculus.

When studying $\mathcal{X}$ as an untyped language, some unexpected special properties surfaced: it became apparent that $\mathcal{X}$ provides an excellent general purpose machine, very well suited to encode various calculi (for details, see [1]). Amongst the calculi studied in that paper, the Calculus of Explicit Substitutions, $\lambda\mathbf{x}$, stands out, for which $\mathcal{X}$ provides a 'subatomic' level by decomposing explicit substitutions into smaller components. Even more, the calculus is symmetric [3]; the '*cut*', represented by $(P\widehat{\alpha} * \widehat{x}Q)$ represents, in a sense, the explicit substitution of P for x in Q, but also that of Q for α in P.

Perhaps the main feature of $\mathcal{X}$ is that it constitutes a *variable* and *substitution*-free method of computation. Rather than having variables like x representing places where terms can be inserted, in $\mathcal{X}$ the symbol x represents a *socket*, to which a term can be *attached* via a *plug* α. The definition of reduction on $\mathcal{X}$ shows nicely how the interaction between the two subtly and gently percolates through the terms.

Also, although the origin of $\mathcal{X}$ is a logic, and one could expect it to be close to LC, it is in fact specified as a *conditional term rewriting system* via rewrite rules that specify how to construct new terms using a term of a certain structure. This observation is the reason for the present paper. The only non-standard aspects in the rewrite rules for $\mathcal{X}$ are a notion of binding, and that it treats *three* different classes of open nodes (for plugs, sockets, and circuits).

Starting from this observation, the decision was made to build an interpreter for $\mathcal{X}$ using term graph rewriting techniques; we set out to build a generic reduction tool for $\mathcal{X}$, aimed at interested researchers in programming languages so they could

familiarise themselves with the calculus. The tool allows users to not only input terms in $\mathcal{X}$, but also terms from LC, using the interpretation of the latter into $\mathcal{X}$ as specified in [1]. This paper reports on the first findings of that effort. The interpreter can be found at

`http://www.doc.ic.ac.uk/~jr200/XTool`

and is in fact part of a highly sophisticated tool, that allows the user almost absolute control over the reduction engine, choosing sub-systems, strategies, and cuts to reduce at will.

Whilst testing an initial implementation of the interpreter, illegal reduction behaviour occurred. Analysis of the problem brought forward that the reduction engine generated 'double bindings' of connectors. By the nature of term graph rewriting, sharing is introduced in the engine not just on the circuits involved, but also on their connectors, where 'binding' edges are added from the binding occurrences to the bound ones. Although sharing in the graph interpretation of the initial term is limited to this 'binder-bound' link, this need no longer be the case after reduction. Not only will circuits be shared, but also connectors can be shared in more ways; the reduction engine in fact generated 'double' binding, i.e. more than one binding edge ending in a single connector node. Binding of connectors is problematic in the context of sharing in the same way as sharing is problematic for abstractions in the context of lambda graphs [13].

To tackle this problem, the solution presented in this paper will introduce auxiliary structure to $\mathcal{X}$-term graphs, and a notion of *rebinding* will be added to the reduction system. The idea is to 'peel off a copy' of the graphs which might get affected by the double binding of connectors. The principle for this method is similar to that of [13], but, as argued in many papers since then, this is too restrictive and can be improved upon. In this paper we will presented a solution that only needs to peel off the structure that 'sits above a connector that is used again', because that causes the conflict. All other sharing can remain intact.

1 The $\mathcal{X}$-calculus

In this section we will give the definition of the $\mathcal{X}$-calculus, as studied in detail in [1]. It features two separate categories of 'connectors', *plug* and *socket*, that act as input and output channels, and are defined without a notion of substitution (implicit or explicit).

Definition 1.1 [Syntax] The circuits of the $\mathcal{X}$-calculus are defined by the following syntax, where $x, y, \ldots$ range over the infinite set of *sockets*, and α, β over the infinite set of *plugs*.

$$P, Q ::= \langle x.\alpha \rangle \mid \widehat{y}P\widehat{\beta}\cdot\alpha \mid P\widehat{\beta}\,[y]\,\widehat{x}Q \mid P\widehat{\alpha}\dagger\widehat{x}Q$$

In this definition, the $\widehat{\cdot}$ symbolises that the socket or plug underneath is bound in the circuit. This leads naturally to the following definition of the notions of *free sockets* $fs(P)$ and *free plugs* $fp(P)$ in a circuit P.

Definition 1.2 The *free sockets* and *free plugs* in a circuit are defined by:

$$\begin{aligned}
\mathit{fs}(\langle x.\alpha\rangle) &= \{x\}\\
\mathit{fs}(\widehat{x}P\widehat{\beta}\cdot\alpha) &= \mathit{fs}(P)\backslash\{x\}\\
\mathit{fs}(P\widehat{\alpha}\,[y]\,\widehat{x}Q) &= \mathit{fs}(P)\cup\{y\}\cup\mathit{fs}(Q)\backslash\{x\}\\
\mathit{fs}(P\widehat{\alpha}\dagger\widehat{x}Q) &= \mathit{fs}(P)\cup\mathit{fs}(Q)\backslash\{x\}
\end{aligned}$$

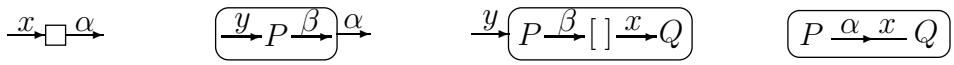
$$\begin{aligned}
\mathit{fp}(\langle x.\alpha\rangle) &= \{\alpha\}\\
\mathit{fp}(\widehat{x}P\widehat{\beta}\cdot\alpha) &= (\mathit{fp}(P)\backslash\{\beta\})\cup\{\alpha\}\\
\mathit{fp}(P\widehat{\alpha}\,[y]\,\widehat{x}Q) &= (\mathit{fp}(P)\backslash\{\alpha\})\cup\mathit{fp}(Q)\\
\mathit{fp}(P\widehat{\alpha}\dagger\widehat{x}Q) &= (\mathit{fp}(P)\backslash\{\alpha\})\cup\mathit{fp}(Q)
\end{aligned}$$

The set of *free connectors* is the union of those: $\mathit{fc}(P) = \mathit{fs}(P)\cup\mathit{fp}(P)$. A connector (socket or plug) which is not free is called *bound*.

We will, as usual, identify processes that only differ in the names of bound connectors (α-conversion).

We use the following terminology for our circuits: $\langle x.\alpha\rangle$ is called a *capsule*, $(\widehat{y}P\widehat{\beta}\cdot\alpha)$ an *export* circuit, $(P\widehat{\beta}\,[y]\,\widehat{x}Q)$ a *mediator*, and $(P\widehat{\alpha}\dagger\widehat{x}Q)$ a *cut*.

Diagrammatically, we represent these circuits as:



The circuits of $\mathcal{X}$ can be seen as a collection of wires, each with an input and an output. When two circuits interact, they do so through *one* input and *one* output name *only*, that are *bound* in the interaction. This interaction can be possible via a circuit like $(P\widehat{\alpha}\dagger\widehat{x}Q)$ that expresses an active computation; it will try to connect the wires ending with α in the circuit called P to the wires beginning with x in the circuit called Q. On the other hand, they can be bound as in $(P\widehat{\beta}\,[y]\,\widehat{x}Q)$, which expresses two circuits that try to connect the wires ending with β and beginning with x, but that need another circuit to mediate between them; this new circuit will need to interact via the input name y (which might appear in P and Q as well). Also, a circuit P that is willing to interact via the connectors y and β can itself be made available (exported) via the name α, as in $(\widehat{y}P\widehat{\beta}\cdot\alpha)$.

At any given moment, all unconnected inputs and outputs in a circuit make up the collection of *free connectors* that are *inactive* during the computational step; the bound connectors are all involved in some interaction.

The calculus, defined by the reduction rules below, explains in detail how cuts are distributed through circuits to be eventually evaluated at the level of capsules. Reduction is defined by giving how the basic syntactic structures that are well-connected interact, and specifies how to deal with propagating active nodes in the computation to points where they can interact.

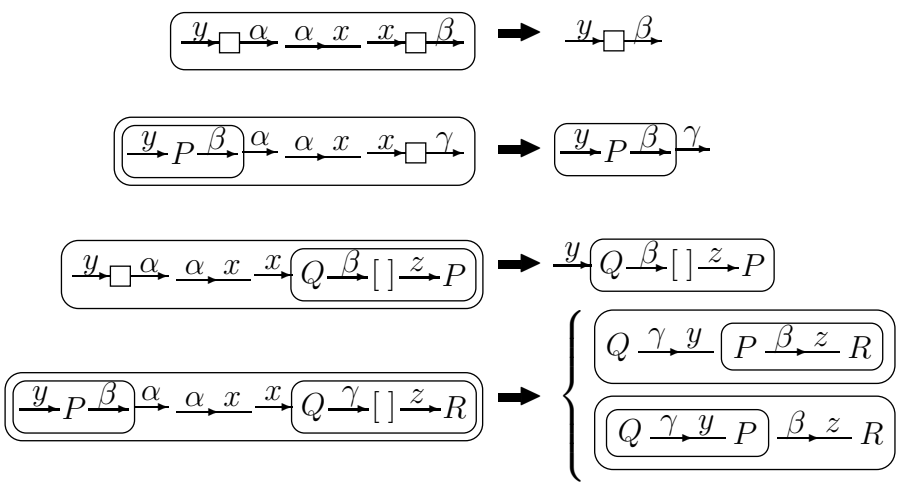


Fig. 1. The logical rules in their diagrammatical representation

Definition 1.3 [Reduction: **Logical rules**]

$$(\textit{cap}): \quad \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle \to \langle y.\beta\rangle$$

$$(\textit{exp}): \quad (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}\langle x.\gamma\rangle \to \widehat{y}P\widehat{\beta}\cdot\gamma, \qquad \alpha \notin \textit{fp}(P)$$

$$(\textit{med}): \quad \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{z}P) \to Q\widehat{\beta}\,[y]\,\widehat{z}P, \qquad x \notin \textit{fs}(Q,P)$$

$$(\textit{ins}): (\widehat{y}P\widehat{\beta}\cdot\alpha)\widehat{\alpha}\dagger\widehat{x}(Q\widehat{\gamma}\,[x]\,\widehat{z}R) \to \begin{cases} Q\widehat{\gamma}\dagger\widehat{y}(P\widehat{\beta}\dagger\widehat{z}R) & \alpha \notin \textit{fp}(P) \\ (Q\widehat{\gamma}\dagger\widehat{y}P)\widehat{\beta}\dagger\widehat{z}R & x \notin \textit{fs}(Q,R) \end{cases}$$

The diagrammatical representation of these rules is given in Figure 1.

The first three logical rules above specify a renaming (reconnecting) procedure, whereas the last rule specifies the basic computational step: it links a function, available over the unique plug α, to an open adjacent position in the mediator circuit that ports the unique socket x.

Notice that these rules are specified as a *conditional term rewriting system*; the non-standard aspects are binding of connectors and that it treats *three* different classes of open nodes (for plugs, sockets, and circuits).

The syntax is extended with two ***flagged,*** or *active* cuts:

$$P ::= \ldots \mid P_1\widehat{\alpha}\not\nearrow\widehat{x}P_2 \mid P_1\widehat{\alpha}\not\nwarrow\widehat{x}P_2$$

Terms constructed from the restricted syntax without those flagged cuts are called *pure* (the diagrammatical representation of flagged cuts is the same as that for unflagged cuts). These flagged cuts are generated when the above logical rules are not applicable, and either reduce to normal cuts when dealing with a capsule, or are propagated through the circuit.

Definition 1.4 [Reduction: **Activating the cuts**] We define the following two activation rules.

$$(\textit{act-}\textsc{l}): P\widehat{\alpha}\dagger\widehat{x}Q \to P\widehat{\alpha}\not\nearrow\widehat{x}Q \textit{ if } P \textit{ does not introduce } \alpha$$

$$(\textit{act-}\textsc{r}): P\widehat{\alpha}\dagger\widehat{x}Q \to P\widehat{\alpha}\not\nwarrow\widehat{x}Q \textit{ if } Q \textit{ does not introduce } x$$

where:

P **introduces** x Either $P = Q\widehat{\alpha}\,[x]\,\widehat{y}R$ and $x \notin \mathit{fs}(Q, R)$, or $P = \langle x.\delta\rangle$.

P **introduces** δ Either $P = \widehat{x}Q\widehat{\beta}\cdot\delta$ and $\delta \notin \mathit{fp}(Q)$, or $P = \langle x.\delta\rangle$.

The activated cuts (notice that activation is only possible of cuts to which the logical rules cannot be applied) are introduced to obtain a fine-tuned reduction system. An activated cut is processed by 'pushing' it, systematically, in the direction indicated by the tilting of the dagger, through the syntactic structure, until a cut is created that involves a capsule. The cut is then deactivated, such that a logical rule can be applied or, else, the 'pushing' can go on, but now in the other direction.

Definition 1.5 [Reduction: **Propagation rules**] **Left propagation**

$$
\begin{array}{lll}
(\mathit{d}\mathrm{L}): & \langle y.\alpha\rangle\widehat{\alpha}\not\leftarrow\widehat{x}P \to \langle y.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}P & \\
(\mathrm{L1}): & \langle y.\beta\rangle\widehat{\alpha}\not\leftarrow\widehat{x}P \to \langle y.\beta\rangle & \beta \neq \alpha \\
(\mathrm{L2}): & (\widehat{y}Q\widehat{\beta}\cdot\alpha)\widehat{\alpha}\not\leftarrow\widehat{x}P \to (\widehat{y}(Q\widehat{\alpha}\not\leftarrow\widehat{x}P)\widehat{\beta}\cdot\gamma)\widehat{\gamma}\dagger\widehat{x}P, & \gamma \textit{ fresh} \\
(\mathrm{L3}): & (\widehat{y}Q\widehat{\beta}\cdot\gamma)\widehat{\alpha}\not\leftarrow\widehat{x}P \to \widehat{y}(Q\widehat{\alpha}\not\leftarrow\widehat{x}P)\widehat{\beta}\cdot\gamma, \gamma \neq \alpha & \\
(\mathrm{L4}): & (Q\widehat{\beta}\,[z]\,\widehat{y}P)\widehat{\alpha}\not\leftarrow\widehat{x}R \to (Q\widehat{\alpha}\not\leftarrow\widehat{x}R)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\not\leftarrow\widehat{x}R) & \\
(\mathrm{L5}): & (Q\widehat{\beta}\dagger\widehat{y}P)\widehat{\alpha}\not\leftarrow\widehat{x}R \to (Q\widehat{\alpha}\not\leftarrow\widehat{x}R)\widehat{\beta}\dagger\widehat{y}(P\widehat{\alpha}\not\leftarrow\widehat{x}R) &
\end{array}
$$

Right propagation

$$
\begin{array}{lll}
(\mathit{d}\mathrm{R}): & P\widehat{\alpha}\not\rightarrow\widehat{x}\langle x.\beta\rangle \to P\widehat{\alpha}\dagger\widehat{x}\langle x.\beta\rangle & \\
(\mathrm{R1}): & P\widehat{\alpha}\not\rightarrow\widehat{x}\langle y.\beta\rangle \to \langle y.\beta\rangle & y \neq x \\
(\mathrm{R2}): & P\widehat{\alpha}\not\rightarrow\widehat{x}(\widehat{y}Q\widehat{\beta}\cdot\gamma) \to \widehat{y}(P\widehat{\alpha}\not\rightarrow\widehat{x}Q)\widehat{\beta}\cdot\gamma & \\
(\mathrm{R3}): & P\widehat{\alpha}\not\rightarrow\widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{y}R) \to P\widehat{\alpha}\dagger\widehat{z}((P\widehat{\alpha}\not\rightarrow\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\not\rightarrow\widehat{x}R)) & z \textit{ fresh} \\
(\mathrm{R4}): & P\widehat{\alpha}\not\rightarrow\widehat{x}(Q\widehat{\beta}\,[z]\,\widehat{y}R) \to (P\widehat{\alpha}\not\rightarrow\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\not\rightarrow\widehat{x}R) & z \neq x \\
(\mathrm{R5}): & P\widehat{\alpha}\not\rightarrow\widehat{x}(Q\widehat{\beta}\dagger\widehat{y}R) \to (P\widehat{\alpha}\not\rightarrow\widehat{x}Q)\widehat{\beta}\dagger\widehat{y}(P\widehat{\alpha}\not\rightarrow\widehat{x}R) &
\end{array}
$$

We write $\to$ for the (reflexive, transitive, compatible) reduction relation induced by these (16) rules.

Intuitively, reduction in $\mathcal{X}$ will connect sockets with plugs. In this view, the cut $P\widehat{\alpha}\dagger\widehat{x}Q$ expresses that connections need to be established between all occurrences of α in P and all occurrences of x in Q. In some cases, these occurrences are unique and at the top (i.e are introduced) and the connection is easy; the logical rules deal with these. If this is not the case, the occurrences of the connectors have to 'be looked for'; the propagation rules deal with this. Left propagation, i.e. running $P\widehat{\alpha}\not\leftarrow\widehat{x}Q$, will have the effect that all occurrences of α in P will be connected to an x in Q; similarly, right propagation, i.e. running $P\widehat{\alpha}\not\rightarrow\widehat{x}Q$, will have the effect that all occurrences of x in Q will be connected to an α in P; notice that these two operations need not have the same effect, especially if there is no α in P or x in Q.

The behaviour of propagation can be broken down and described as happening in two stages. First, the cut propagates in one direction until a circuit with

which it can connect is encountered, i.e. one exhibiting a connector at the outermost level. This is the case formally described by rules (L2) and (R3). If this circuit is non-trivial (has sub-circuits), two things need to happen: first of all, all (possible) occurrences of the exhibited connector need to be dealt with, and this is done by propagating to all sub-circuits; secondly, a fresh cut is deposited that will deal with the one remaining occurrence. Since that last occurrence can now be assumed to be unique (all other will be dealt with by the propagation), the resulting structure might now be a logical cut; for this reason, the additional cut is not active. In a sense, in this second stage of propagation, the fresh cut re-evaluates its 'situation': if a logical rule is applicable the cut can be eliminated, otherwise, it is propagated in the opposite direction (it will not propagate in the same direction, since there the connector is now introduced.

In rule (L2) and (R3), we rename to avoid α (x) to occur both bound and free; but, in fact, no confusion is possible, so the α-conversion here is almost cosmetic.

Since the reduction relation $\rightarrow$ on $\mathcal{X}$ mimics logical cut-elimination (successfully), it is not confluent; this comes in fact from the critical pair that activates a cut $P\widehat{\alpha}\dagger\widehat{x}Q$ in two ways if P does not introduce α and Q does not introduce x. When activating according to the first criterion, the reduction will connect all α in P with the connectors x in Q; if α does not appear in P, this will return P. When using the second, the reduction will connect all x in Q with the connectors α in P; if x does not appear in Q, this will return Q. As an example, consider that (when $\beta \neq \gamma$, and $y \neq z$)

$$(\widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\dagger\widehat{z}\langle y.\delta\rangle \rightarrow \begin{cases} (\widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\nwarrow\widehat{z}\langle y.\delta\rangle \rightarrow \langle y.\delta\rangle \\ (\widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma)\widehat{\beta}\nearrow\widehat{z}\langle y.\delta\rangle \rightarrow \widehat{x}\langle x.\alpha\rangle\widehat{\alpha}\cdot\gamma \end{cases}$$

So, if activation in both directions is possible, the end result of the reduction can be different.

In [1], two strategies were introduced which explicitly favour one kind of activation whenever the above critical pair occurs:

Definition 1.6 [Call-by-name and Call-by-value]

- The CBV strategy only allows activation of a cut via (*act*-L) when it could be activated in two ways; we write $P \rightarrow_{\mathrm{V}} Q$ in that case.
- The CBN strategy only allows activation of such a cut via (*act*-R); we then write $P \rightarrow_{\mathrm{N}} Q$.

The justification of the terminology is given by Theorem 2.3.

In [1] some basic properties were shown, which essentially show that the calculus is well-behaved, as well as the relation between $\mathcal{X}$ and a number of other calculi. These results motivate the formulation of new rules:

Lemma 1.7 ([1]) *The following reduction rules are admissible:*

$$\begin{aligned}
(gc\text{-L}): &\quad P\widehat{\alpha}\not\swarrow\widehat{x}Q \to P && \text{if } \alpha \notin fp(P), P \text{ pure}\\
(gc\text{-R}): &\quad P\widehat{\alpha}\not\searrow\widehat{x}Q \to Q && \text{if } x \notin fs(Q), Q \text{ pure}\\
(ren\text{-L}): &\quad P\widehat{\delta}\dagger\widehat{z}\langle z.\alpha\rangle \to P[\alpha/\delta] && P \text{ pure}\\
(ren\text{-R}): &\quad \langle z.\alpha\rangle\widehat{\alpha}\dagger\widehat{x}P \to P[z/x] && P \text{ pure}
\end{aligned}$$

The latter two could be introduced into the reduction engine for $\mathcal{X}$ as *renaming rules*. The effect of this is like α-conversion, since we must rename *all* occurrences of the binding connector throughout the whole term, although the binding occurrence itself disappears. There is a danger to these rules, however, since they in fact perform a *destructive update* on the graph that represents P. This is sound only if we can guarantee that there are no other references to P. This problem is again much the same as in the case of lambda graphs, where an abstraction involved in two applications has to be copied. The problem could perhaps be avoided by keeping track of reference counts while reducing, or perhaps by an abstract, compile-time analysis. much in the spirit of uniqueness types [7]. In view of this difficulty, however, the renaming rules are, for the moment, omitted from the tool.

2 The relation with the Lambda Calculus

In this section, we will briefly highlight the relation between LC and $\mathcal{X}$. We assume the reader to be familiar with LC [4]; we just recall the definition of lambda terms and β-contraction.

Definition 2.1 [Lambda terms and β-contraction [4]]

(i) The set Λ of *lambda terms* is defined by the syntax:

$$M ::= x \mid \lambda x.M \mid M_1M_2$$

(ii) The reduction relation $\to_\beta$ is defined as the contextual (i.e. compatible [4]) closure of the rule:

$$(\lambda x.M)N \to_\beta M[N/x]$$

The relation $\twoheadrightarrow_\beta$ is the reflexive and transitive closure of $\to_\beta$, and the $=_\beta$ is the equivalence relation generated by $\twoheadrightarrow_\beta$

We now define the direct encoding of λ-terms into $\mathcal{X}$.

Definition 2.2 [[1]] The interpretation of λ-terms into $\mathcal{X}$ via the plug α, $[\![M]\!]_\alpha$, is defined by:

$$\begin{aligned}
[\![x]\!]_\alpha &= \langle x.\alpha\rangle\\
[\![\lambda x.M]\!]_\alpha &= \widehat{x}[\![M]\!]_\beta\widehat{\beta}\cdot\alpha, && \beta \text{ fresh}\\
[\![MN]\!]_\alpha &= [\![M]\!]_\gamma\widehat{\gamma}\dagger\widehat{x}([\![N]\!]_\beta\widehat{\beta}\,[x]\,\widehat{y}\langle y.\alpha\rangle), && x, y, \beta, \gamma \text{ fresh}
\end{aligned}$$

$$
\begin{array}{ll}
\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket_\alpha & = \\
\llbracket\lambda x.xx\rrbracket_\beta\widehat{\beta}\dagger\widehat{z}(\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\ [z]\ \widehat{v}\langle v.\alpha\rangle) & = \\
(\widehat{x}\llbracket xx\rrbracket_\epsilon\widehat{\epsilon}\cdot\beta)\widehat{\beta}\dagger\widehat{z}(\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\ [z]\ \widehat{v}\langle v.\alpha\rangle) & \to (\textit{ins}) \\
\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\dagger\widehat{x}(\llbracket xx\rrbracket_\epsilon\widehat{\epsilon}\dagger\widehat{v}\langle v.\alpha\rangle) & = \\
\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\dagger\widehat{x}((\langle x.\gamma\rangle\widehat{\gamma}\dagger\widehat{z}(\langle x.\delta\rangle\widehat{\delta}\ [z]\ \widehat{w}\langle w.\epsilon\rangle))\widehat{\epsilon}\dagger\widehat{v}\langle v.\alpha\rangle) & \\
 & \to (\textit{ren}\text{-L}), (\textit{ren}\text{-R}), (\textit{act}\text{-R}) \\
\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\curlywedge\widehat{x}(\langle x.\delta\rangle\widehat{\delta}\ [x]\ \widehat{w}\langle w.\alpha\rangle) & \to (\text{R3}) \\
\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\dagger\widehat{z}((\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\curlywedge\widehat{x}\langle x.\delta\rangle)\widehat{\delta}\ [z]\ \widehat{w}(\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\curlywedge\widehat{x}\langle w.\alpha\rangle)) & \\
 & \to (\textit{d}\text{R}), (\textit{ren}\text{-R}), (\text{R1}) \\
\llbracket\lambda y.y\rrbracket_\gamma\widehat{\gamma}\dagger\widehat{z}(\llbracket\lambda y.y\rrbracket_\delta\widehat{\delta}\ [z]\ \widehat{w}\langle w.\alpha\rangle) & = \\
(\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\gamma)\widehat{\gamma}\dagger\widehat{z}(\llbracket\lambda y.y\rrbracket_\delta\widehat{\delta}\ [z]\ \widehat{w}\langle w.\alpha\rangle) & \to (\textit{ins}) \\
\llbracket\lambda y.y\rrbracket_\delta\widehat{\delta}\dagger\widehat{y}(\langle y.\mu\rangle\widehat{\mu}\dagger\widehat{w}\langle w.\alpha\rangle) & \to (\textit{cap}), (\textit{ren}\text{-R}) \\
\llbracket\lambda y.y\rrbracket_\alpha &
\end{array}
$$

Fig. 2. (Symbolically) reducing $\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket_\alpha$ to $\llbracket\lambda y.y\rrbracket_\alpha$.

Notice that every sub-term of $\llbracket M\rrbracket_\alpha$ has exactly one free plug.

In Figure 2 we show how the process $\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket_\alpha$ can be reduced, using the rules for $\mathcal{X}$ presented above.

In [1], the following relation is shown between (call-by-name, call-by-value) reduction in LC and $\mathcal{X}$:

Theorem 2.3 ([1]) (i) *If* $M \to_\beta N$, *then* $\llbracket M\rrbracket_\alpha \to \llbracket N\rrbracket_\alpha$.
(ii) *If* $M \to_{\text{V}} N$ *then* $\llbracket M\rrbracket_\gamma \to_{\text{V}} \llbracket N\rrbracket_\gamma$.
(iii) *If* $M \to_{\text{N}} N$ *then* $\llbracket M\rrbracket_\gamma \to_{\text{N}} \llbracket N\rrbracket_\gamma$.

The last two results link the concept of '*name*' and '*value*' quite nicely to $\mathcal{X}$: the circuits that can be called a *value* in $\mathcal{X}$ are those that introduce a plug, and a *name* is a circuit that introduces a socket. However, notice that, in contrast to LC, in $\mathcal{X}$ the CBV reduction is not a sub-system of the CBN reduction; perhaps another notion than 'name' should be used for the latter. Notice that, for the notion of reduction $\to_{\text{V}}$ on $\mathcal{X}$, in the cut $P\widehat{\alpha}\dagger\widehat{x}Q$, the cut is only right-activated if Q does not introduce x, and P is a value. So, P is only 'inserted' into Q if it is a value, which makes this reduction justifiably called 'call-by-value'.

The converse of the results of Theorem 2.3 do not hold a-priori: this is mainly because the reduction relation in $\mathcal{X}$ is far more complex than just those reductions between (interpretations of) lambda terms, and it could be that there exists a path between $\llbracket M\rrbracket_\alpha$ and $\llbracket N\rrbracket_\alpha$ which does not correspond to a LC-reduction path between M and N.

3 A Term Graph Rewriting System for $\mathcal{X}$

The implementation of an interpreter for $\mathcal{X}$ could of course have been done in the traditional way, via a string-reduction machine. The choice to use Term Graph Rewriting instead is motivated mainly from the observation that the reduction rules for $\mathcal{X}$ form a conditional term rewriting system; for a general term rewriting system, the term graph technology provides the best platform for implementation.

This resulted in a tool encapsulating an interpreter, as can be found at

`http://www.doc.ic.ac.uk/~jr200/XTool`.

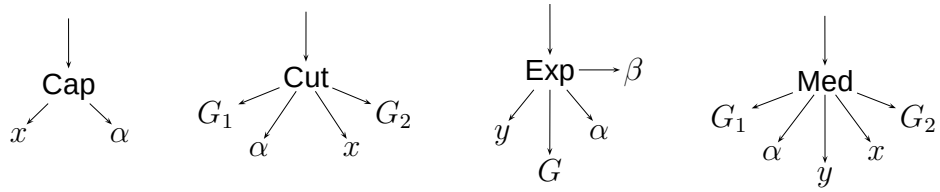
This tool is in fact highly sophisticated, allowing for much more than just reducing circuits; it allows the user almost absolute control over the reduction engine, extending the set of rules, choosing sub-systems, strategies, and cuts to reduce at will. It provides an elegant means of exploring $\mathcal{X}$, and completely by-passes the need to run circuits 'by hand'. The elegance of the final product was achieved not only through thorough engineering, but also through the beauty of the TGRS formalism, which allows for a high-level specification and understanding of the reduction engine that was needed for running $\mathcal{X}$.

The technique applied is the standard *match*, *build*, *link*, *re-direct*, and *garbage collection* approach of [11,5,6], where terms and rewrite rules are *lifted* to graphs. By the process of lifting, the connectors appear only once in the generated graph, which immediately introduces sharing. Rewrite rules also become graphs with *two* sub-graphs that each possess a root, and are united via shared leaves.

Definition 3.1 [Term graphs] The $\mathcal{X}$ term graphs are defined by the following syntax:

$$G ::= \mathsf{Cap}(x, \alpha) \mid \mathsf{Cut}(G_1, \alpha, x, G_2) \mid \mathsf{Exp}(y, G, \alpha, \beta) \mid \mathsf{Med}(G_1, \alpha, y, x, G_2)$$

which corresponds to the (ordered) graphs



(notice that we do not need to use the hat, since it is immediately clear which connectors are the binding occurrences)

Definition 3.2 [Graph interpretation] For each term M, the *graph interpretation of* M, $[\![M]\!]_{\mathrm{G}}$, (using the notation of [8]) is defined in Figure 3.

Definition 3.3 The lifting of the $\mathcal{X}$ reduction rules to term graph rewriting rules is now expressed by:

$$[\![\mathit{left} \to \mathit{right}]\!]_{\mathrm{G}} = \langle r_l \mid G_l \cup G_r \rangle$$
$$\text{where } \langle r_l \mid G_l \rangle = [\![\mathit{left}]\!]_{\mathrm{G}}$$
$$\langle r_r \mid G_r \rangle = [\![\mathit{right}]\!]_{\mathrm{G}}$$

$$
\begin{aligned}
\llbracket x \rrbracket_{\mathrm{G}} &= \langle x \mid \emptyset \rangle \\
\llbracket \alpha \rrbracket_{\mathrm{G}} &= \langle \alpha \mid \emptyset \rangle \\
\llbracket \langle x.\alpha \rangle \rrbracket_{\mathrm{G}} &= \langle r \mid \{r = \mathsf{Cap}(r_1, r_2)\} \cup G_1 \cup G_2 \rangle, \\
&\quad \text{where } \langle r_1 \mid G_1 \rangle = \llbracket x \rrbracket_{\mathrm{G}}, \\
&\qquad \langle r_2 \mid G_2 \rangle = \llbracket \alpha \rrbracket_{\mathrm{G}}, \\
\llbracket M\widehat{\alpha} \dagger \widehat{x}N \rrbracket_{\mathrm{G}} &= \langle r \mid \{r = \mathsf{Cut}(r_1, r_2, r_3, r_4)\} \cup G_1 \cup G_2 \cup G_3 \cup G_4 \rangle \\
&\quad \text{where } \langle r_1 \mid G_1 \rangle = \llbracket M \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_2 \mid G_2 \rangle = \llbracket \alpha \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_3 \mid G_3 \rangle = \llbracket x \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_4 \mid G_4 \rangle = \llbracket N \rrbracket_{\mathrm{G}} \\
\llbracket \widehat{y}M\widehat{\alpha} \cdot \beta \rrbracket_{\mathrm{G}} &= \langle r \mid \{r = \mathsf{Exp}(r_1, r_2, r_3, r_4)\} \cup G_1 \cup G_2 \cup G_3 \cup G_4 \rangle \\
&\quad \text{where } \langle r_1 \mid G_1 \rangle = \llbracket y \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_2 \mid G_2 \rangle = \llbracket M \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_3 \mid G_3 \rangle = \llbracket \beta \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_4 \mid G_4 \rangle = \llbracket \alpha \rrbracket_{\mathrm{G}} \\
\llbracket M\widehat{\alpha}\,[y]\,\widehat{x}N \rrbracket_{\mathrm{G}} &= \langle r \mid \{r = \mathsf{Med}(r_1, r_2, r_3, r_4, r_5)\} \cup G_1 \cup G_2 \cup G_3 \cup G_4 \cup G_5 \rangle \\
&\quad \text{where } \langle r_1 \mid G_1 \rangle = \llbracket M \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_2 \mid G_2 \rangle = \llbracket \alpha \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_3 \mid G_3 \rangle = \llbracket y \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_4 \mid G_4 \rangle = \llbracket x \rrbracket_{\mathrm{G}} \\
&\qquad \langle r_5 \mid G_5 \rangle = \llbracket N \rrbracket_{\mathrm{G}}
\end{aligned}
$$

Fig. 3. Graph interpretation of $\mathcal{X}$ terms

Using this interpretation, the term graph rules that correspond to $(\mathrm{L2})$ and $(\mathrm{R3})$ are as in Figure 4 and 5 (writing n rather than r_n).

Of course, this 'semantics' of interpreting terms and reduction rules in our (extended) TGRS needs to be formally checked. For this, we need to define a notion of *unravelling*:

Definition 3.4 *Unrav*(G), the *unravelling* of a $\mathcal{X}$-term graph G is obtained by traversing the graph top-down (notice that we have no cyclic structures), and copying, for all shared graphs, all nodes in that graph that are not free connectors.

This definition depends clearly on the *in-degree*, the number of arcs going into a node. This notion has not been included in the definition of graphs we use here, but that could easily be amended.

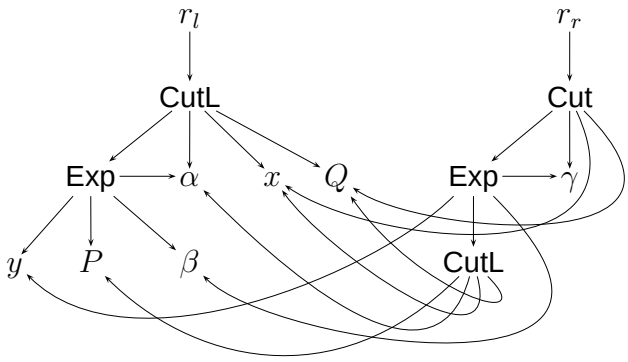
Using this notion, we can now prove the following adequacy result:

Theorem 3.5 *Let G_1, G_2 be $\mathcal{X}$-term graphs, and P_1, P_2 be $\mathcal{X}$-circuits such that Unrav$(G_i) = \llbracket P_i \rrbracket_{\mathrm{G}}$, for $i = 1, 2$. If $G_1 \to G_2$ using the term graph rules, then $P_1 \to P_2$. Moreover, if G_2 is in normal form, then so is P_2.*

Proof. *It is easy to show that there exists a G_3 such that $\llbracket P_2 \rrbracket_{\mathrm{G}} =$ Unrav(G_3) and*

$(\text{L2}) : (\widehat{y}P\widehat{\beta}{\cdot}\alpha)\widehat{\alpha}\not\prec\widehat{x}Q \to (\widehat{y}(P\widehat{\alpha}\not\prec\widehat{x}Q)\widehat{\beta}{\cdot}\gamma)\widehat{\gamma}\dagger\widehat{x}Q,\ \gamma$ *fresh*

$$
\begin{array}{rl}
\langle r_l \mid \{r_l & = \mathsf{CutL}(1,2,3,4) \\
1 & = \mathsf{Exp}(5,6,7,2) \\
2 & = \alpha \\
3 & = x \\
4 & = Q \\
5 & = y \\
6 & = P \\
7 & = \beta \\
r_r & = \mathsf{Cut}(8,9,3,4) \\
8 & = \mathsf{Exp}(5,10,7,9) \\
9 & = \gamma \\
10 & = \mathsf{CutL}(6,2,3,4)\rangle
\end{array}
$$



$$\mathsf{CutL}(\mathsf{Exp}(5{:}y, 6{:}P, 7{:}\beta, 2{:}\alpha), 2, 3{:}x, 4{:}Q) \to \mathsf{Cut}(\mathsf{Exp}(5, \mathsf{CutL}(6,2,3,4), 7, 9{:}\gamma), 9, 3, 4) \quad \gamma \textit{ fresh}$$

Fig. 4. Various term graph representations of rule (L2).

$[\![P_1]\!]_{\text{G}} \to G_3$. *This reduction induces a reduction from* P_1 *to* P_2. *If* G_2 *contains no cuts, then neither does* $[\![P_2]\!]_{\text{G}}$, *nor* P_2. □

In fact, the term graph rewriting system defined above is set up to satisfy this property. The converse does not hold, by the very nature of sharing in the term graph rewriting engine. However, the following partial soundness result should hold:

Corollary 3.6 *If* $P \to Q$, *then there exists* R *such that* $Q \to R$ *and* $[\![P]\!]_{\text{G}} \to [\![R]\!]_{\text{G}}$.

4 Implementation issues

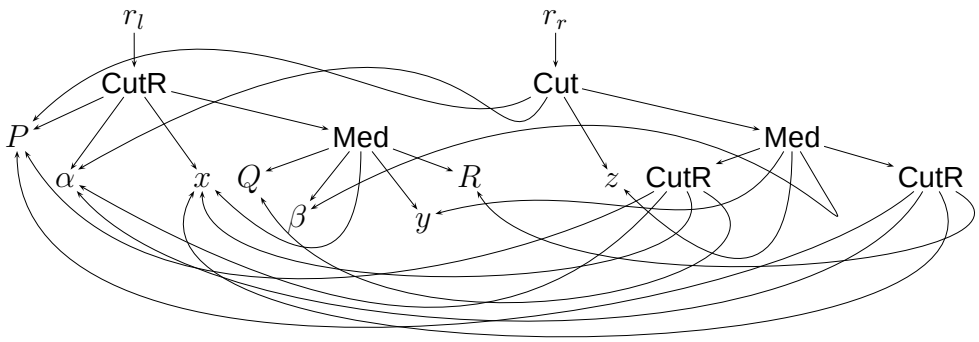
Functional strategies for $\mathcal{X}$ were found to be implementable via a strict first match policy on an ordered list of the rules in the TGRS. As suggested by Definition 1.6, the Call-by-Name strategy (CBN) was implemented by an ordering of the rules which saw $(\textit{act}\text{-R})$ appear before $(\textit{act}\text{-L})$, whilst the Call-by-Value strategy (CBV) was implemented by giving priority to $(\textit{act}\text{-L})$. In the scenario that a cut could be activated in either way, this policy ensures that the first rule encountered is applied.

Implementing sharing in the TGRS rewrite engine required little extra attention, since node redirection involved only pointer updates and an increment/decrement of reference counts; a string-reduction machine would have involved implementation of an internal copying mechanism for terms. Additionally, there was never a need to search term-graphs for locations of bound connectors, since any bindings of nodes would be shared.

The TGRS approach proved to be a good learning tool for the calculus, allowing users to visualise each reduction step. The user can clearly see cuts being propag-

$(\text{R3}) : P\widehat{\alpha}\nwarrow\widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{y}R) \to P\widehat{\alpha}\dagger\widehat{z}((P\widehat{\alpha}\nwarrow\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\nwarrow\widehat{x}R))$, x *fresh*

$$\begin{array}{ll} \langle r_l \mid \{r_l = \mathsf{CutR}(1,2,3,4) & 7 = y \\ 1 = P & 8 = R \\ 2 = \alpha & 9 = z \\ 3 = x & r_r = \mathsf{Cut}(1,2,9,10) \\ 4 = \mathsf{Med}(5,6,3,7,8) & 10 = \mathsf{Med}(11,6,7,9,12) \\ 5 = Q & 11 = \mathsf{CutR}(1,2,3,5) \\ 6 = \beta & 12 = \mathsf{CutR}(1,2,3,8)\rangle \end{array}$$



$$\mathsf{CutR}(1{:}P, 2{:}\alpha, 3{:}x, \mathsf{Med}(5{:}Q, 6{:}\beta, 3, 7{:}y, 8{:}R)) \to \mathsf{Cut}(1, 2, 9{:}z, \mathsf{Med}(\mathsf{CutR}(1,2,3,5), 6, 9, 7, \mathsf{CutR}(1,2,3,8)))$$

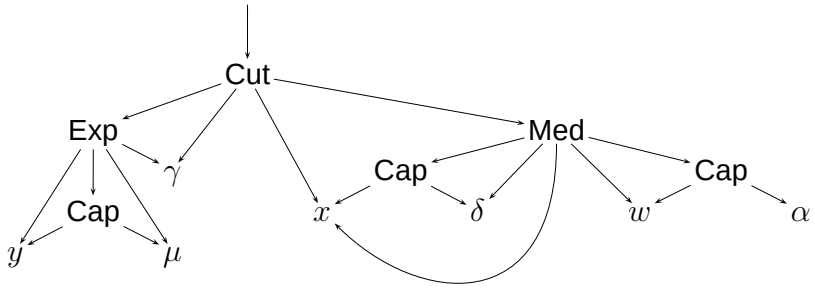
Fig. 5. Various term graph representations of rule (R3).

ated through to sub-graphs, and alpha-conversion taking place via the rebinding nodes. The main problem that needed to be solved in this implementation was the fact that $\mathcal{X}$ knows bound connectors. Similarly to the case for LC [13], binding is considered problematic in the context of sharing, so, when interpreting terms and rewrite rules as graphs, some care has to be taken.

Example 4.1 Take the following circuit

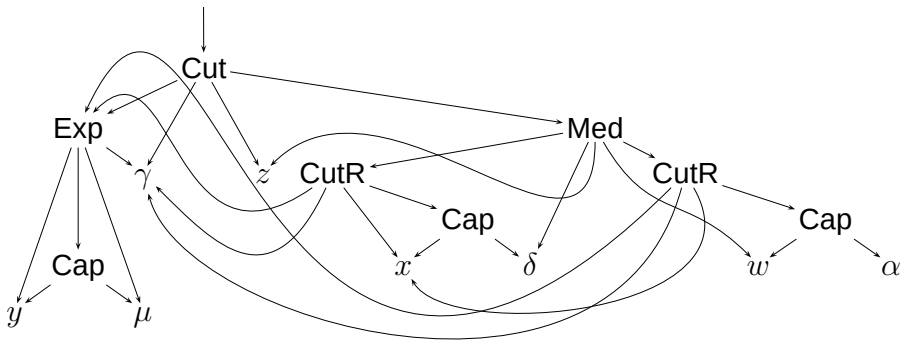
$$(\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\gamma)\widehat{\gamma}\dagger\widehat{x}(\langle x.\delta\rangle\widehat{\delta}\,[x]\,\widehat{w}\langle w.\alpha\rangle)$$

The graph that represents this circuit is



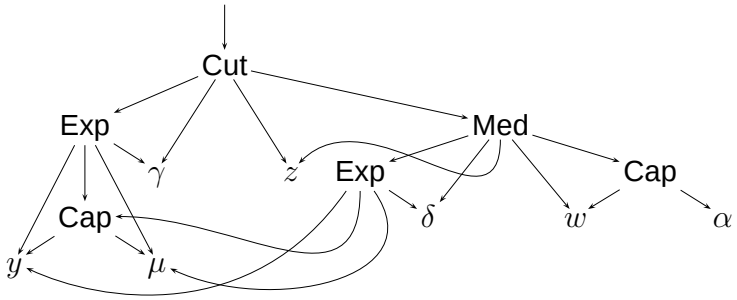
Applying the term graph rules (*act*-R) and (R3) will generate

$$(\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\gamma)\widehat{\gamma}\dagger\widehat{z}(((\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\gamma)\widehat{\gamma}\nwarrow\widehat{x}\langle x.\delta\rangle)\widehat{\delta}\,[z]\,\widehat{w}((\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\gamma)\widehat{\gamma}\nwarrow\widehat{x}\langle w.\alpha\rangle))$$



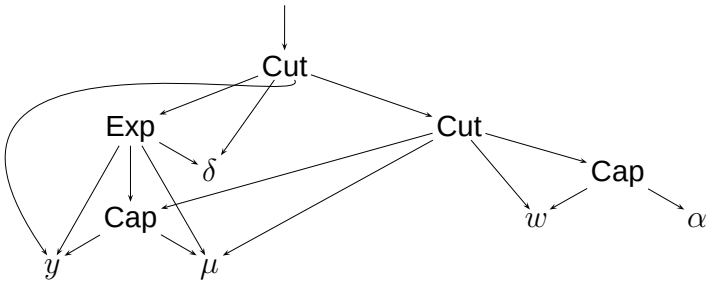
Applying the term graph rules $(d\text{R})$, $(\textit{exp})$, and (R1) will generate

$$(\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\gamma)\widehat{\gamma}\,\dagger\,\widehat{z}((\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\delta)\widehat{\delta}\;[z]\;\widehat{w}\langle w.\alpha\rangle)$$



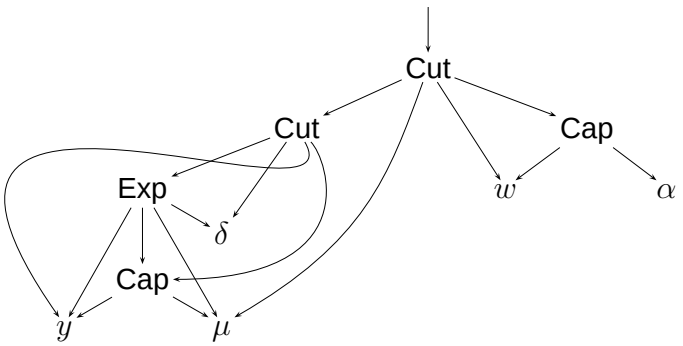
As is clear from this graph, the capsule on the left is now shared, and there are *two* binders to both y and μ, coming from both export nodes. Continuing the execution of this graph (via the first variant of rule $(\textit{ins})$) yields the graph

$$(\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\delta)\widehat{\delta}\,\dagger\,\widehat{y}(\langle y.\mu\rangle\widehat{\mu}\,\dagger\,\widehat{w}\langle w.\alpha\rangle)$$



Notice that δ is introduced in this graph, whereas y is not, so this graph reduces to

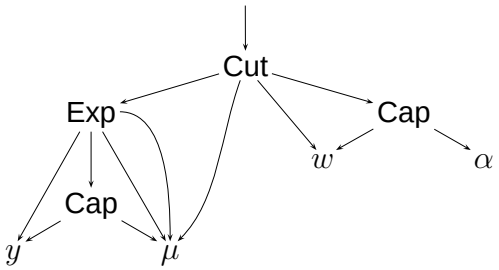
$$((\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\delta)\widehat{\delta}\,\dagger\,\widehat{y}\langle y.\mu\rangle)\widehat{\mu}\,\dagger\,\widehat{w}\langle w.\alpha\rangle$$



which, by the way, is what we would have obtained from the graph above had we applied the *second* variant of rule (ins).

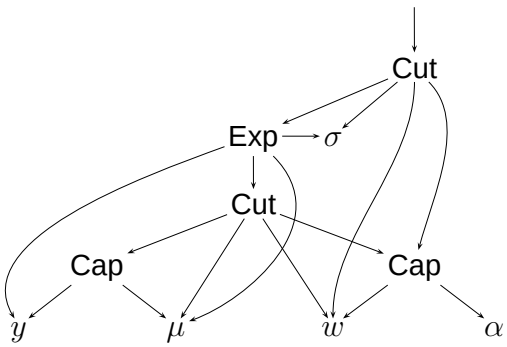
Notice that now there are two nested binders to μ. We can assume that the innermost binds the strongest, so the left free μ is bound by the innermost, and the right free μ is bound by the outermost, but this does not solve the conflict here. If we now first reduce the inner cut using (exp), we obtain

$$(\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\mu)\widehat{\mu}\,\dagger\,\widehat{w}\langle w.\alpha\rangle$$

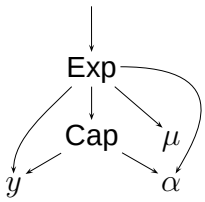


Notice that the right-most free μ is bound by the outermost binder, but the rewrite rules will generate the wrong graph. To reduce the cut, we first check if μ is introduced; this is not so, since $\mu \in \mathit{fp}(\mathsf{Cap}(y,\mu))$, thus denying the intended application of the rule (exp). Instead, we propagate the cut and obtain, using rules $(\mathit{act}\text{-}\mathrm{L})$, $(\mathrm{L2})$ and $(\mathit{d}\mathrm{L})$,

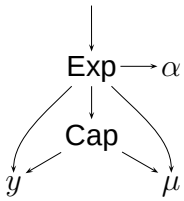
$$(\widehat{y}(\langle y.\mu\rangle\widehat{\mu}\,\dagger\,\widehat{w}\langle w.\alpha\rangle)\widehat{\mu}\cdot\sigma)\widehat{\sigma}\,\dagger\,\widehat{w}\langle w.\alpha\rangle$$



which reduces by rule (cap) and (exp) to $\widehat{y}\langle y.\alpha\rangle\widehat{\mu}\cdot\alpha$.



This is not the correct result; we should have obtained $\widehat{y}\langle y.\mu\rangle\widehat{\mu}\cdot\alpha$.



Evidently, just sharing nodes indiscriminately creates problems: for example, since both y and β are bound in $\widehat{y}P\widehat{\beta}\cdot\alpha$, it should not be permitted that both are bound *again* in $\widehat{y}(P\widehat{\alpha}\not\frown\widehat{x}Q)\widehat{\beta}\cdot\gamma$, since connecting them can happen only *once*. Although the root node of $\widehat{y}P\widehat{\beta}\cdot\alpha$ is not reachable from the root of the right-hand side, by the very nature of term graph rewriting, the first might be shared, so there might be another entry into that node.

Potentially, this sharing of binders introduces the problem that P now contains one or more βs bound under both an export term and a left-cut term at the same time. The propagation rules in our calculus define ways in which terms can travel inside each other, and so we leave ourselves open to the case when the export term and the propagating left-cut meet.

Example 4.2 Figure 6 shows a reduction sequence (an alternative to the one of Figure 2) which clearly illustrates this conflict. Step 6 of the reduction shows β bound in an export term and a cut. When we propagate the cut $\hat{\beta}\dagger\hat{h}$ through to the left term, it encounters the export term $\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta$, which also shares β as its binder (Step 9). If we were reducing the term by hand, we could perhaps perform an α-conversion step, but what we need is a solution on the level of the rewrite system.

Notice that the occurrences of each β in the term represent the same node in the term graph, and so it is difficult to make this distinction without adding extra conditions to our rewrite rules. Application of rule $(\textsc{l}2)$ swaps the innermost binder (export) and outermost binder (left-cut) in the term graph, leading to an incorrect reduction sequence.

This example, along with others which propagate shared binders, prompts us to define the rewrite rules in such a way that binders are never shared directly in a term. To ensure this, we extend our TGRS definition with a notion of rebinding by introducing additional rewrite rules. The original idea was to copy the graphs which might get affected by the double binding of connectors, as also done in [13].

(1) $\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket_\gamma \rightarrow (\mathit{ins}{-}\mathrm{L}), (\mathit{ren}\text{-}\mathrm{L})$

(2) $\llbracket(xx)\langle x = \lambda y.y\rangle\rrbracket_\gamma \rightarrow (\mathit{med}), (\mathit{act}\text{-}\mathrm{R}), (\mathrm{R3}), (\mathrm{R1}), (\mathit{d}\mathrm{R}), (\mathit{exp})$

(3) $(\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\psi)\widehat{\psi} \dagger \widehat{k}((\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta)\widehat{\delta}\ [k]\ \widehat{h}\langle h.\gamma\rangle)$
$\rightarrow (\mathit{ins}L)$

(4) $(\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta)\widehat{\delta} \dagger \widehat{y}(\langle y.\beta\rangle\widehat{\beta} \dagger \widehat{h}\langle h.\gamma\rangle) \rightarrow (\mathit{act}\text{-}\mathrm{R})$

(5) $(\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta)\widehat{\delta} \nwarrow \widehat{y}(\langle y.\beta\rangle\widehat{\beta} \dagger \widehat{h}\langle h.\gamma\rangle) \rightarrow (\mathrm{R5})$

(6) $((\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta)\widehat{\delta} \nwarrow \widehat{y}\langle y.\beta\rangle)\widehat{\beta} \dagger \widehat{h}((\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta)\widehat{\delta} \nwarrow \widehat{y}\langle h.\gamma\rangle)$
$\rightarrow (\mathrm{R1}), (\mathit{d}\mathrm{R}), (\mathit{act}\text{-}\mathrm{L})$

(7) $((\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta)\widehat{\delta} \dagger \widehat{y}\langle y.\beta\rangle)\widehat{\beta} \nearrow \widehat{h}\langle h.\gamma\rangle \rightarrow (\mathrm{L5})$

(8) $((\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta)\widehat{\beta} \nearrow \widehat{h}\langle h.\gamma\rangle)\widehat{\delta} \dagger \widehat{y}(\langle y.\beta\rangle\widehat{\beta} \nearrow \widehat{h}\langle h.\gamma\rangle)$
$\rightarrow (\mathit{d}\mathrm{L}), (\mathit{cap})$

(9) $((\widehat{y}\langle y.\beta\rangle\widehat{\beta}\cdot\delta)\widehat{\beta} \nearrow \widehat{h}\langle h.\gamma\rangle)\widehat{\delta} \dagger \widehat{y}\langle y.\gamma\rangle \rightarrow (\mathrm{L3})$

(10) $(\widehat{y}(\langle y.\beta\rangle\widehat{\beta} \nearrow \widehat{h}\langle h.\gamma\rangle)\widehat{\beta}\cdot\delta)\widehat{\delta} \dagger \widehat{y}\langle y.\gamma\rangle \rightarrow (\mathit{d}\mathrm{L}), (\mathit{cap})$

(11) $(\widehat{y}\langle y.\gamma\rangle\widehat{\beta}\cdot\delta)\widehat{\delta} \dagger \widehat{y}\langle y.\gamma\rangle \rightarrow (\mathit{exp})$

(12) $\widehat{y}\langle y.\gamma\rangle\widehat{\beta}\cdot\gamma$

Fig. 6. Reducing $\llbracket(\lambda x.xx)(\lambda y.y)\rrbracket_\gamma$ to $\widehat{y}\langle y.\gamma\rangle\widehat{\beta}\cdot\gamma$.

Our approach is to only peel off the structure that 'sits above a connector that is used again', because that causes the conflict. All other sharing can stay intact.

This results in the (term graph) definition of rebinding a socket (**rs**) as given in Figure 7 (the first rule has also a variant that deals with the case $v = x$). These rename the doubly bound connector by, essentially, copying that structure of a graph which contains that binder whilst introducing the new connector, thereby destroying the sharing of the connector via binding edges.

The function **rp**(M, α, β) as given in Figure 8, is defined to build a new graph G' where the free occurrences of α in G are replaced with β and any binders encountered *in* G are made fresh. Since this is, essentially, a copying function, when we move the rebinding mechanism *under* binders, as in the case RbSExpPG, we would create double binders for those bound connectors we have just passed. Therefore, we need to rebind those as well.

Definition 4.3 The function **rs**, in linear graph notation, is defined in Figure 7; the function **rp** is defined in Figure 8, as graphs.

The functions **rs** and **rp** are expressed as normal term graph rewriting rules, and are in fact part of the reduction engine in our tool. By placing them in the right position in the list of rules, we can cause the copying function they perform to be done *eagerly* of *lazily*. The latter is accomplished by putting these rules at the very

$$
\begin{array}{ll}
(\text{RbSCapRen}): & \mathbf{rs}(\mathsf{Cap}(1{:}x, 2{:}\alpha), 1, 3{:}y) \to \mathsf{Cap}(3, 2) \\
(\text{RbSCapIgnore}): & \mathbf{rs}(1{:}\mathsf{Cap}(z, \alpha), x, y) \to 1 \\
(\text{RbSExpPG}): & \mathbf{rs}(\mathsf{Exp}(1{:}z, 2{:}M, 3{:}\beta, 4{:}\alpha), 5{:}x, 6{:}y) \to \\
 & \quad \mathsf{Exp}(7{:}k, \mathbf{rs}(\mathbf{rp}(\mathbf{rs}(2, 5, 6), 3, 8{:}\delta), 1, 7), 8, 4) \\
(\text{RbSMedRen}): & \mathbf{rs}(\mathsf{Med}(1{:}M, 2{:}\alpha, 3{:}x, 4{:}z, 5{:}N), 3, 6{:}y) \to \\
 & \quad \mathbf{rs}(\mathsf{Med}(1, 2, 6, 4, 5), 3, 6) \\
(\text{RbSMedPG}): & \mathbf{rs}(\mathsf{Med}(1{:}M, 2{:}\alpha, 3{:}k, 4{:}z, 5{:}N), 6{:}x, 7{:}y) \to \\
 & \quad \mathsf{Med}(\mathbf{rp}(\mathbf{rs}(1, 6, 7), 2, 9{:}\beta), 9, 3, 10{:}k, \mathbf{rs}(\mathbf{rs}(5, 6, 7), 4, 10)) \\
(\text{RbSCutPG}): & \mathbf{rs}(\mathsf{Cut}(1{:}M, 2{:}\alpha, 3{:}z, 4{:}N), 5{:}x, 6{:}y) \to \\
 & \quad \mathsf{Cut}(\mathbf{rp}(\mathbf{rs}(1, 5, 6), 2, 7{:}\beta), 7, 8{:}k, \mathbf{rs}(\mathbf{rs}(4, 5, 6), 3, 8)) \\
(\text{RbSCutLPG}): & \mathbf{rs}(\mathsf{CutL}(1{:}M, 2{:}\alpha, 3{:}z, 4{:}N), 5{:}x, 6{:}y) \to \\
 & \quad \mathsf{CutL}(\mathbf{rp}(\mathbf{rs}(1, 5, 6), 2, 7{:}\beta), 7, 8{:}k, \mathbf{rs}(\mathbf{rs}(4, 5, 6), 3, 8)) \\
(\text{RbSCutLPG}): & \mathbf{rs}(\mathsf{CutR}(1{:}M, 2{:}\alpha, 3{:}z, 4{:}N), 5{:}x, 6{:}y) \to \\
 & \quad \mathsf{CutR}(\mathbf{rp}(\mathbf{rs}(1, 5, 6), 2, 7{:}\beta), 7, 8{:}k, \mathbf{rs}(\mathbf{rs}(4, 5, 6), 3, 8))
\end{array}
$$

Fig. 7. The function **rs**

bottom of the list. This will have the effect that rebinding only takes place if no other rule matches. In this way, although we destroy sharing of nodes, we only do so when necessary, preserving the parallel reduction character of our engine as much as possible.

Using the functions **rs** and **rp** gives a different formal definition of interpreting terms in $\mathcal{X}$ as graphs. The term graph representation of each (improved, because dealing with rebinding) rule is quite involved. As suggested by the examples above (Examples 4.1 and 4.2), term rewrite rules which introduce sharing of binders need to copy these in order to avoid the conflict. We give the full definition below in linear notation.

Definition 4.4 [TGRS Rewrite Rules] **Logical rules**

$$
\begin{array}{rl}
(\textit{cap}): & \mathsf{Cut}(\mathsf{Cap}(1{:}y, 2{:}\alpha), 2, x{:}3, \mathsf{Cap}(3, 4{:}\beta)) \to \mathsf{Cap}(1, 4) \\
(\textit{exp}): & \mathsf{Cut}(\mathsf{Exp}(1{:}y, 2{:}P, 3{:}\beta, 4{:}\alpha), 4, x{:}5, \mathsf{Cap}(5, 6{:}\gamma)) \to \mathsf{Exp}(1, 2, 3, 6) \\
(\textit{med}): & \mathsf{Cut}(\mathsf{Cap}(1{:}y, 2{:}\alpha), 2, x{:}3, \mathsf{Med}(4{:}Q, 5{:}\beta, 3, 6{:}z, 7{:}P))) \to \\
 & \quad \mathsf{Med}(4, 5, 1, 6, 7) \\
(\textit{ins}\text{-R}): & \mathsf{Cut}(\mathsf{Exp}(1{:}y, 2{:}N, 3{:}\beta, 4{:}\alpha), 4), 5{:}x, \mathsf{Med}(6{:}Q, 7{:}\gamma, 5, 8{:}z, 9{:}P))) \to \\
 & \quad \mathsf{Cut}(6, 7, 1, \mathsf{Cut}(2, 3, 8, 9)) \\
(\textit{ins}\text{-L}): & \mathsf{Cut}(\mathsf{Exp}(1{:}y, 2{:}N, 3{:}\beta, 4{:}\alpha), 4), 5{:}x, \mathsf{Med}(6{:}Q, 7{:}\gamma, 5, 8{:}z, 9{:}P))) \to \\
 & \quad \mathsf{Cut}(\mathsf{Cut}(6, 7, 1, 2), 3, 8, 9)
\end{array}
$$

Cut Activation

$$
\begin{array}{rl}
(\textit{act}\text{-L}): & \mathsf{Cut}(1{:}P, 2{:}\alpha, 3{:}x, 4{:}Q) \to \mathsf{CutL}(1, 2, 3, 4) \\
(\textit{act}\text{-R}): & \mathsf{Cut}(1{:}P, 2{:}\alpha, 3{:}x, 4{:}Q) \to \mathsf{CutR}(1, 2, 3, 4)
\end{array}
$$

Left propagation

$(d\text{L})$: $\mathsf{CutL}(1{:}\mathsf{Cap}(y, 2{:}\alpha), 2, 3{:}x, 4{:}P) \rightarrow \mathsf{Cut}(1, 2, 3, 4)$

(L1) : $\mathsf{CutL}(1{:}\mathsf{Cap}(y, \beta), \alpha, x, P) \rightarrow 1$

(L2) : $\mathsf{CutL}(\mathsf{Exp}(1{:}y, 2{:}P, 3{:}\beta, 4{:}\alpha), 4, 5{:}x, 6{:}Q) \rightarrow$
$\mathsf{Cut}(\mathsf{Exp}(1, \mathsf{CutL}(2, 4, 5, 6), 3, 7{:}\gamma), 7, 8{:}z, \mathbf{rs}\,(6, 5, 8))$

(L3) : $\mathsf{CutL}(\mathsf{Exp}(1{:}y, 2{:}P, 3{:}\beta, 4{:}\alpha), 4, 5{:}x, 6{:}Q) \rightarrow$
$\mathsf{Exp}(1, \mathsf{CutL}(2, 4, 5, 6), 3, 4)$

(L4) : $\mathsf{CutL}(\mathsf{Med}(1{:}Q, 2{:}\beta, 3{:}z, 4{:}y, 5{:}N), 6{:}\alpha, 7{:}x, 8{:}P)) \rightarrow$
$\mathsf{Med}(\mathsf{CutL}(1, 6, 7, 8), 2, 3, 4, \mathsf{CutL}(5, 6, k{:}9, \mathbf{rs}\,(8, 7, 9))$

(L5) : $\mathsf{CutL}(\mathsf{Cut}(1{:}Q, 2{:}\beta, 3{:}y, 4{:}N), 5{:}\alpha, 6{:}x, 7{:}P)) \rightarrow$
$\mathsf{Cut}(\mathsf{CutL}(1, 5, 6, 7), 2, 3, \mathsf{CutL}(4, 5, 8{:}k, \mathbf{rs}\,(7, 6, 8)))$

Right propagation

$(d\text{R})$: $\mathsf{CutR}(1{:}P, 2{:}\alpha, 3{:}x, 4{:}\mathsf{Cap}(3, \beta)) \rightarrow \mathsf{Cut}(1, 2, 3, 4)$

(R1) : $\mathsf{CutR}(P, \alpha, x{:}1, 2{:}\mathsf{Cap}(1, \beta)) \rightarrow 2$

(R2) : $\mathsf{CutR}(1{:}P, 2{:}\alpha, 3{:}x, \mathsf{Exp}(4{:}y, 5{:}Q, 6{:}\beta, 7{:}\gamma)) \rightarrow$
$\mathsf{Exp}(4, \mathsf{CutR}(1, 2, 3, 5), 6, 7)$

(R3) : $\mathsf{CutR}(1{:}P, 2{:}\alpha, 3{:}x, \mathsf{Med}(4{:}Q, 5{:}\beta, 3, 6{:}y, 7{:}N) \rightarrow$
$\mathsf{Cut}(1, 2, 8{:}z, \mathsf{Med}(\mathsf{CutR}(\mathbf{rp}\,(1, 2, 9{:}\mu), 9, 3, 4), 5, 8, 6,$
$\mathsf{CutR}(\mathbf{rp}\,(1, 2, 10{:}\eta), 10, 3, 7)))$

(R4) : $\mathsf{CutR}(1{:}P, 2{:}\alpha, 3{:}x, \mathsf{Med}(4{:}Q, 5{:}\beta, 6{:}z, 7{:}y, 8{:}N) \rightarrow$
$\mathsf{Med}(\mathsf{CutR}(1, 2, 3, 4), 5, 6, 7, \mathsf{CutR}(\mathbf{rp}\,(1, 2, 9{:}\gamma), 9, 3, 8))$

(R5) : $\mathsf{CutR}(1{:}P, 2{:}\alpha, 3{:}x, \mathsf{Cut}(4{:}Q, 5{:}\beta, 6{:}y, 7{:}N) \rightarrow$
$\mathsf{Cut}(\mathsf{CutR}(1, 2, 3, 4), 5, 6, \mathsf{CutR}(\mathbf{rp}\,(1, 2, 8{:}\gamma), 8, 3, 7))$

Now rules (L2) and (R3)

(L2) : $(\widehat{y}P\widehat{\beta}{\cdot}\alpha)\widehat{\alpha}\not\leftarrow\widehat{x}Q \rightarrow (\widehat{y}(P\widehat{\alpha}\not\leftarrow\widehat{x}Q)\widehat{\beta}{\cdot}\gamma)\widehat{\gamma}\dagger\widehat{x}Q,\ \gamma$ *fresh*

(R3) : $P\widehat{\alpha}\not\rightarrow\widehat{x}(Q\widehat{\beta}\,[x]\,\widehat{y}R) \rightarrow P\widehat{\alpha}\dagger\widehat{z}((P\widehat{\alpha}\not\rightarrow\widehat{x}Q)\widehat{\beta}\,[z]\,\widehat{y}(P\widehat{\alpha}\not\rightarrow\widehat{x}R)),\ x$ *fresh*

will be interpreted by the term graph rules in Figure 9. Notice that the call to the function **rs** builds a version of P that uses a fresh socket z to connect rather than x. Evaluating the rebinding rules builds a version of P with fresh binder names. This ensures there is only ever one pointer to nodes that bind over P or the local binders in P.

In Rule (R5) the α node is rebound to γ. Once again, this is to ensure no two sub-terms can share common binders.

Future work

We want to add a notion of type assignment to both $\mathcal{X}$ and its term graph implementation; although the first seems straightforward, the second most likely will

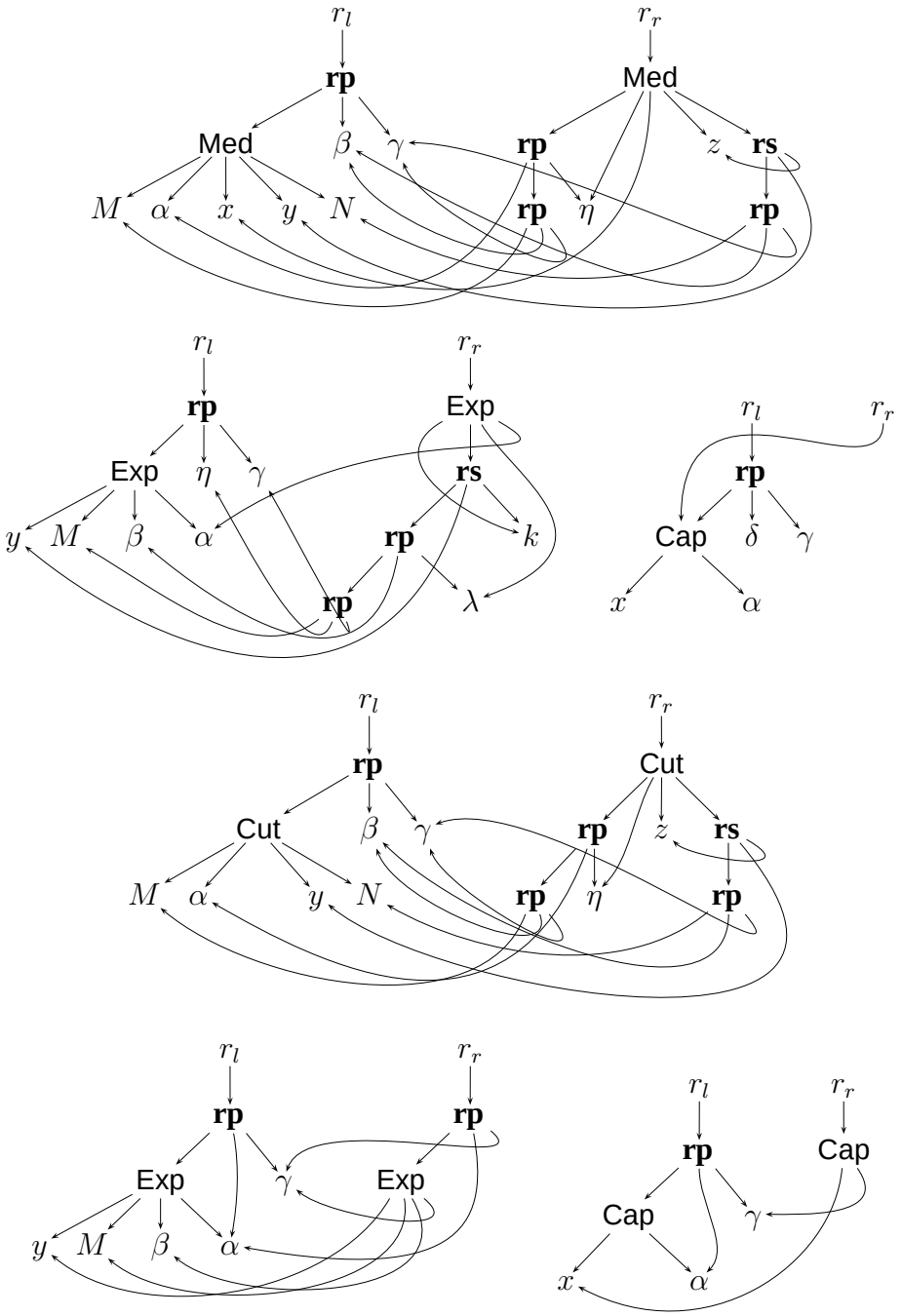


Fig. 8. The function **rp**

need intersection types, and need to be defined with care.

Since the completion of this paper, and its presentation in Rome, we have found another approach to the rewriting problem solved here. In [2] we view the problem not as one on the level of graph rewriting, as done here, but on the level of $\mathcal{X}$ itself, where it turns out to be that of α-conversion. That paper discusses a number of different solutions for that problem.

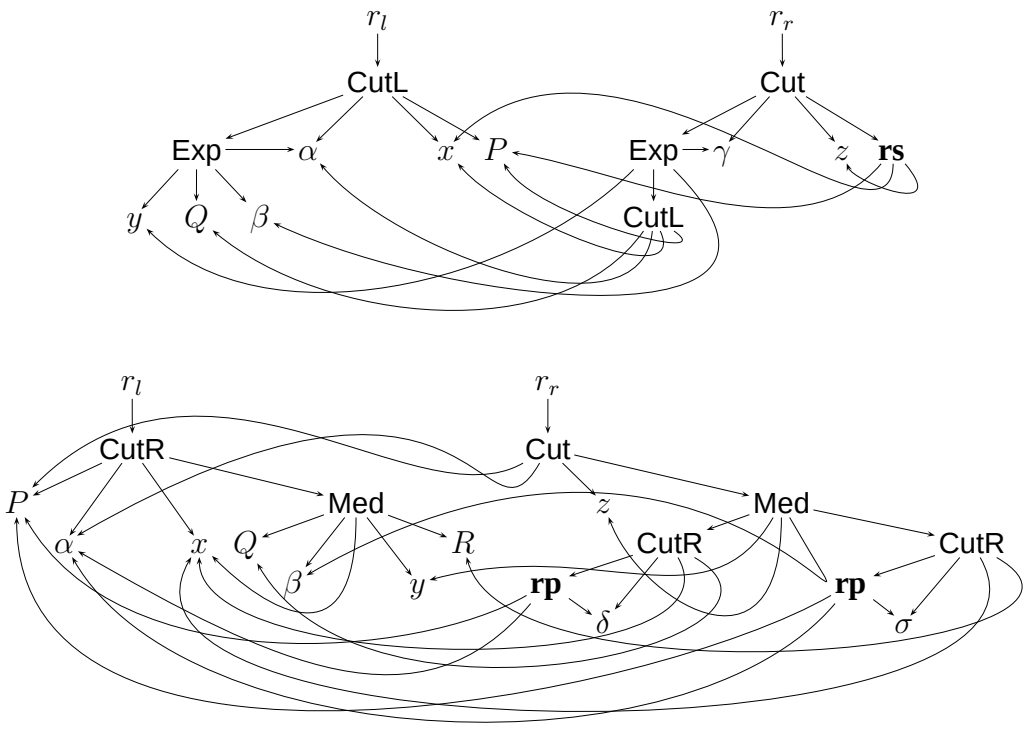


Fig. 9. Rules (L2) and (R3) as term graph rewriting rules with rebinding.

References

[1] S. van Bakel, S. Lengrand, and P. Lescanne. The language $\mathcal{X}$: circuits, computations and classical logic. *Submitted*, 2004.

[2] S. van Bakel, J. Raghunandan, and A. Summers. Term Graphs, α-conversion and Principal Types for $\mathcal{X}$. *Submitted*, 2005.

[3] F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Information and Computation*, 125(2):103–117, 1996.

[4] H. Barendregt. *The Lambda Calculus: its Syntax and Semantics*. North-Holland, Amsterdam, revised edition, 1984.

[5] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, and M.R. Sleep. Term graph rewriting. In *Proceedings of PARLE, Parallel Architectures and Languages Europe*, Eindhoven, The Netherlands, volume 259-II of *Lecture Notes in Computer Science*, pages 141–158. Springer-Verlag, 1987.

[6] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, and M.R. Sleep. Towards an Intermediate Language based on Graph Rewriting. In *Proceedings of PARLE, Parallel Architectures and Languages Europe*, Eindhoven, The Netherlands, volume 259-II of *Lecture Notes in Computer Science*, pages 159–175. Springer-Verlag, 1987.

[7] E. Barendsen and S. Smetsers. Conventional and Uniqueness Typing in Graph Rewrite Systems. In R.K. Shyamasunda, editor, *Proceedings of FST&TCS '93*.

*13*th *Conference on Foundations of Software Technology and Theoretical Computer Science*, Bombay, India, volume 761 of *Lecture Notes in Computer Science*, pages 41–52. Springer-Verlag, 1993.

[8] E. Barendsen and S. Smetsers. Conventional and Uniqueness Typing in Graph Rewrite Systems. *Mathematical Structures of Computer Science*, 1996.

[9] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the 5 th ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, pages 233–243. ACM, 2000.

[10] Stéphane Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In Bernhard Gramlich and Salvador Lucas, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.

[11] R. Sleep, M.J. Plasmeijer, and M.C.J.C van Eekelen, editors. *Term Graph Rewriting. Theory and Practice*. Wiley, 1993.

[12] Christian Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, October 2000.

[13] C.P. Wadsworth. *Semantics and pragmatics of the lambda calculus*. PhD thesis, Oxford University, 1971. Thesis CST-33-85.